# BDA_Lab3

Olayemi Morrison (olamo208) and Greeshma Jeev Koothuparambil(greko370)

2024-05-16

## Assignment

Implement in Spark (PySpark) a kernel model to predict the hourly temperatures for a date and place in Sweden. To do so, you should use the files temperature-readings.csv and stations.csv from previous labs. Specifically, the forecast should consist of the predicted temperatures from 4 am to 24 pm in an interval of 2 hours for a date and place in Sweden.

```
from __future__ import division
from math import radians, cos, sin, asin, sqrt, exp
from datetime import datetime
from pyspark import SparkContext, StorageLevel

sc = SparkContext(appName = "BDALab3")

def haversine(lon1, lat1, lon2, lat2):
    """
    Calculate the great circle distance between two points
    on the earth (specified in decimal degrees)
    """
    # convert decimal degrees to radians
    lon1, lat1, lon2, lat2 = map(radians, [lon1, lat1, lon2, lat2])
    # haversine formula
    dlon = lon2 - lon1
    dlat = lat2 - lat1
    a = sin(dlat/2)**2 + cos(lat1) * cos(lat2) * sin(dlon/2)**2
    c = 2 * asin(sqrt(a))
    km = 6367 * c
    return km

def d_date(date1,date2):
    """
    Calculate the distance between two day
    """
    date_diff=abs((date2 - date1).days)
    return date_diff

def d_time(time1,time2):
    """
    Calculate the distance between two time
    """
    time_diff=abs((time2 - time1).total_seconds() / 3600)
    return time_diff
```

```
def sum_kernel(d_distance,d_date,d_time):
    """
    Calculate the sum of the kernel
    """
    distance_kernel=exp(-(d_distance**2)/(h_distance**2))
    time_kernel=exp(-(d_time**2)/(h_time**2))
    date_kernel=exp(-(d_date**2)/(h_date**2))

    return distance_kernel+time_kernel+date_kernel

h_distance = 480
h_date = 6
h_time = 3
a = 58.4274 # Up to you
b = 14.826 # Up to you
date = "2014-11-04" # Up to you
DATE_TIMESTAMP=datetime.strptime(date,"%Y-%m-%d")

stations = sc.textFile("BDA/input/stations.csv")
temps = sc.textFile("BDA/input/temperature-readings.csv")

# Broadcast
# Station data: station_num, distance
stations_data=stations.map(lambda line: line.split(";")).
map(lambda x:(str(x[0]),haversine(b,a,float(x[4]),float(x[3]))))
bc=sc.broadcast(stations_data.collectAsMap())

joined_data = temps.sample(False, 0.1).map(lambda line: line.split(";"))\
                .map(lambda x:(str(x[0]),(datetime.strptime(x[1],"%Y-%m-%d"),
                datetime.strptime(x[2],"%H:%M:%S"),float(x[3]))))\
                .filter(lambda x: (x[1][0]<=DATE_TIMESTAMP))\
                .map(lambda x: (x[0],(bc.value[x[0]],d_date(x[1][0],
                DATE_TIMESTAMP),x[1][1],x[1][2])))

joined_data.cache()
```

Use a kernel that is the sum of three Gaussian kernels: -The first to account for the distance from a station to the point of interest. -The second to account for the distance between the day a temperature measurement was made and the day of interest. -The third to account for the distance between the hour of the day a temperature measurement was made and the hour of interest. Choose an appropriate smoothing coefficient or width for each of the three kernels above. You do not need to use cross-validation.

```
h_distance = 480
h_date = 6
h_time = 3
a = 58.4274 # Up to you
b = 14.826 # Up to you
date = "2014-11-04" # Up to you

y_sum=[]

for time in ["0:00:00", "22:00:00", "20:00:00", "18:00:00", "16:00:00", "14:00:00",
"12:00:00", "10:00:00", "08:00:00", "06:00:00", "04:00:00"]:
    time=datetime.strptime(time,"%H:%M:%S")
```

```
    sum_kernel_rdd=joined_data.filter(lambda x:(x[1][1]>0 or
    (x[1][2] == 0 and  x[1][2]<time)))\
    .map(lambda x:(sum_kernel(x[1][0],x[1][1],d_time(time,x[1][2])),x[1][3]))
    sum_kernel_rdd=sum_kernel_rdd.map(lambda x:(x[0],x[0]*x[1]))
    .reduce(lambda x,y:(x[0]+y[0],x[1]+y[1]))
    y_sum.append(sum_kernel_rdd[1]/sum_kernel_rdd[0])

time_list=["0:00:00", "22:00:00", "20:00:00", "18:00:00", "16:00:00", "14:00:00",
"12:00:00", "10:00:00", "08:00:00", "06:00:00", "04:00:00"]
dictionary = {k: v for k, v in zip(time_list, y_sum)}

dictionary_rdd = parallelize(dictionary.items())
lines_rdd = dictionary_rdd.map(lamda x: "{}\t{}".format(x[0],x[1]))
lines_rdd.saveAsTextFile("BDA/output/predictions")
```

**Result**

0:00:00 5.576996866734825
22:00:00 5.750164300184201
20:00:00 5.7391265024573
18:00:00 5.872109182398333
16:00:00 6.087360188896718
14:00:00 6.256528584965731
12:00:00 6.256148350713531
10:00:00 6.054364603682301
08:00:00 5.68502337277898
06:00:00 5.365450282322555
04:00:00 5.275377563480532

## Question 1

Show that your choice for the kernels' width is sensible, i.e. it gives more weight to closer points. Discuss why your definition of closeness is reasonable.

**Answer**

The choice of kernel widths (h_distance = 480, h_date = 6, h_time = 3) is sensible because it aligns with observed temperature variations across distance, date, and time. The Gaussian kernel's exponential weighting ensures closer points, whether in distance, date, or time, have more influence, which is crucial for accurate temperature prediction. This approach mirrors the real-world influence of proximity on weather conditions, making the definitions of closeness both practical and reasonable.

A bandwidth of 480 km ensures that points closer than 480 km have a significant influence, while points further away have exponentially less influence due to the nature of the Gaussian kernel.

A 6-day bandwidth ensures that dates closer to the reference date (e.g., May 18) are given more weight, with influence decreasing for dates further away.

A bandwidth of 3 hours ensures that times closer to each other (within a few hours) are given more weight, while times further apart have less influence.

## Question 2

Repeat the exercise using a kernel that is the product of the three Gaussian kernels above. Compare the results with those obtained for the additive kernel. If they differ, explain why.

```
y_prod=[]

for time in ["0:00:00", "22:00:00", "20:00:00", "18:00:00", "16:00:00", "14:00:00",
"12:00:00", "10:00:00", "08:00:00", "06:00:00", "04:00:00"]:
    time=datetime.strptime(time,"%H:%M:%S")
    prod_kernel_rdd=joined_data.filter(lambda x:(x[1][1]>0 or (x[1][2] == 0 and  x[1][2]<time)))\
        .map(lambda x:(prod_kernel(x[1][0],x[1][1],d_time(time,x[1][2])),x[1][3]))
    prod_kernel_rdd=prod_kernel_rdd.map(lambda x:(x[0],x[0]*x[1])).
    reduce(lambda x,y:(x[0]+y[0],x[1]+y[1]))
    y_prod.append(prod_kernel_rdd[1]/prod_kernel_rdd[0])

time_list=["0:00:00", "22:00:00", "20:00:00", "18:00:00", "16:00:00", "14:00:00",
"12:00:00", "10:00:00", "08:00:00", "06:00:00", "04:00:00"]
dictionary = {k: v for k, v in zip(time_list, y_prod)}

dictionary_rdd = parallelize(dictionary.items())
lines_rdd = dictionary_rdd.map(lamda x: "{}\t{}".format(x[0],x[1]))
lines_rdd.saveAsTextFile("BDA/output/predictions")
```

**Result**

0:00:00 7.604741613862872
22:00:00 7.653980502590221
20:00:00 7.724486100677334
18:00:00 8.001946526185064
16:00:00 8.483044005659343
14:00:00 8.983860394055593
12:00:00 9.062340343106362
10:00:00 8.470913493959479
08:00:00 7.497029388153619
06:00:00 6.933763720447497
04:00:00 7.027176259560958

**Answer**

The results from the product kernel are consistently higher than those from the additive kernel. This discrepancy arises from the fundamental differences in how these kernels aggregate the influence of distance, date, and time.

The product kernel provides higher weights for points that are close in all three dimensions (distance, date, time), leading to higher temperature predictions overall.

This approach is better when the impact of each factor should multiply together strongly. On the other hand, the additive kernel provides a more balanced approach, which might be preferred if we want to ensure that no single component overly dominates the weighting.